

## RabbitMQ and Redis

📄 link tham khảo

<https://www.rabbitmq.com/tutorials/tutorial-one-python.html> <https://www.rabbitmq.com/tutorials/tutorial-one-python.html>  
<https://redis.io/>

Chào các bạn học viên! Rất vui được gặp các bạn trong buổi chia sẻ hôm nay.

Tôi đã nhận tài liệu của các bạn về **RabbitMQ** (cụ thể là Tutorial 1: "Hello World") và **Redis** (trang chủ [Redis - The Real-time Data Platform](#)). Với tư cách là một giảng viên công nghệ, tôi sẽ tổng hợp lại hai công nghệ này một cách **đễ hiểu nhất**, đi từ bản chất vấn đề đến thực hành.

Hãy tưởng tượng chúng ta đang xây dựng một hệ thống phần mềm lớn (như Shopee hay Facebook). Hai vấn đề lớn nhất chúng ta gặp phải là: **Giao tiếp giữa các bộ phận** (làm sao để gửi đơn hàng từ kho sang giao vận mà không bị tắc nghẽn?) và **Tốc độ truy xuất dữ liệu** (làm sao để hiển thị sản phẩm nhanh nhất?).

Đó chính là lý do RabbitMQ và Redis ra đời.

## PHẦN 1: RABBITMQ - "NGƯỜI ĐƯA THƯ TẬN TỤY"

### 1. Tổng quan & Cơ chế (Theo RabbitMQ Tutorial 1)

RabbitMQ là một **Message Broker** (Môi giới tin nhắn).

Hãy tưởng tượng RabbitMQ giống như một **Bưu điện**.

- Khi bạn muốn gửi thư (dữ liệu), bạn bỏ vào hòm thư. Bạn không cần quan tâm bác đưa thư đi xe gì, đi đường nào, bạn chỉ cần biết thư sẽ đến tay người nhận.
- RabbitMQ nhận tin nhắn, lưu trữ nó và chuyển nó đi.

#### Các thành phần chính:

1. **Producer (P)**: Người gửi thư (Chương trình gửi dữ liệu).
2. **Queue (Q)**: Hòm thư (Nơi chứa tin nhắn). Nó nằm bên trong RabbitMQ. Nó giống như một bộ đệm khổng lồ.
3. **Consumer (C)**: Người nhận thư (Chương trình nhận và xử lý dữ liệu).

#### Cơ chế hoạt động:

Producer gửi tin nhắn vào Queue → RabbitMQ giữ tin nhắn ở đó → Consumer rảnh tay sẽ lấy tin nhắn từ Queue ra để xử lý.

→ Điều này giúp hệ thống "Bất đồng bộ" (Asynchronous). Producer cứ gửi, không cần chờ Consumer xử lý xong.

### 2. Code Demo (Python - Sử dụng thư viện pika)

Dựa trên tài liệu bạn gửi, đây là ví dụ "Hello World" kinh điển.

**Cài đặt:** pip install pika

File 1: send.py (Người gửi)

```

1 import pika
2
3 # 1. Kết nối đến RabbitMQ (giả sử chạy ở localhost)
4 connection =
5     pika.BlockingConnection(pika.ConnectionParameters('localhost'))
6     channel = connection.channel()
7
8 # 2. Khai báo hàng đợi tên là 'hello'
9 # (Phải khai báo để đảm bảo nó tồn tại)
10 channel.queue_declare(queue='hello')
11
12 # 3. Gửi tin nhắn
13 message = 'Xin chào RabbitMQ!'
14 channel.basic_publish(exchange='',
15                       routing_key='hello', # Tên hàng đợi muốn gửi vào
16                       body=message)
17 print(f" [x] Đã gửi '{message}'")
18
19 # 4. Đóng kết nối
20 connection.close()

```

File 2: receive.py (Người nhận)

```

1 import pika, sys, os
2
3 def main():
4     # 1. Kết nối
5     connection =
6         pika.BlockingConnection(pika.ConnectionParameters('localhost'))
7         channel = connection.channel()
8
9     # 2. Khai báo lại hàng đợi (để chắc chắn nó tồn tại nếu chạy
10     # receive trước send)
11     channel.queue_declare(queue='hello')
12
13     # 3. Định nghĩa hàm xử lý khi nhận được tin nhắn
14     def callback(ch, method, properties, body):
15         print(f" [x] Đã nhận được: {body.decode()}")
16
17     # 4. Đăng ký nhận tin
18     channel.basic_consume(queue='hello',
19                           on_message_callback=callback,
20                           auto_ack=True)
21
22     print(' [*] Đang chờ tin nhắn. Nhấn CTRL+C để thoát')
23     channel.start_consuming()
24
25 if __name__ == '__main__':
26     try:
27         main()
28     except KeyboardInterrupt:
29         print('Interrupted')
30         try:
31             sys.exit(0)
32         except SystemExit:
33             os._exit(0)

```

### 3. RabbitMQ dùng khi nào?

- **Xử lý nền (Background Tasks):** Người dùng upload file Excel nặng 1GB để xử lý. Bạn không thể bắt họ chờ loading. Bạn gửi yêu cầu vào RabbitMQ, một Worker ngầm sẽ lấy ra xử lý dần.
- **Đảm bảo tin cậy:** Cần chắc chắn tin nhắn không bị mất (RabbitMQ hỗ trợ cơ chế Acknowledge - báo đã nhận).

- **Hệ thống phân tán (Microservices):** Service A muốn nói chuyện với Service B mà không muốn phụ thuộc trực tiếp (Decoupling).

## PHẦN 2: REDIS - "CUỐN SỔ TAY TỐC ĐỘ CAO"

### 1. Tổng quan & Cơ chế (Theo [Redis - The Real-time Data Platform](#) )

Redis (Remote Dictionary Server) không phải là bưu điện, nó giống như một **tờ giấy nháp** hay **bộ nhớ tạm (RAM)** nhưng cực kỳ thông minh.

Nó là một kho lưu trữ cấu trúc dữ liệu trong bộ nhớ (In-memory data store).

#### Đặc điểm cốt lõi:

- **Lưu trên RAM:** Tốc độ cực nhanh (phản hồi dưới 1 mili-giây). Nhanh hơn nhiều so với Database lưu trên ổ cứng (như MySQL/PostgreSQL).
- **Key-Value:** Dữ liệu lưu dạng cặp Khóa - Giá trị.
- **Đa năng:** Dùng làm Database, Cache (bộ nhớ đệm), hoặc Message Broker.

### 2. Code Demo (Python - Sử dụng thư viện redis)

**Cài đặt:** pip install redis

Kịch bản: Lưu Cache (Bộ nhớ đệm)

Giả sử việc tính toán  $a + b$  tốn rất nhiều thời gian, ta sẽ lưu kết quả vào Redis.

```
1 import redis
2 import time
3
4 # 1. Kết nối Redis
5 r = redis.Redis(host='localhost', port=6379, decode_responses=True)
6
7 def lay_thong_tin_san_pham(product_id):
8     # Bước 1: Kiểm tra xem trong Redis có chưa
9     cached_data = r.get(product_id)
10
11     if cached_data:
12         print("> Lấy từ Redis (Siêu nhanh!)")
13         return cached_data
14
15     # Bước 2: Nếu chưa có, giả vờ truy vấn DB (tốn 3 giây)
16     print("🐢 Đang truy vấn Database (Chậm...)...")
17     time.sleep(3)
18     du_lieu_tu_db = f"Thông tin sản phẩm {product_id}"
19
20     # Bước 3: Lưu vào Redis để lần sau dùng (hết hạn sau 10 giây)
21     r.set(product_id, du_lieu_tu_db, ex=10)
22
23     return du_lieu_tu_db
24
25 # Chạy thử
26 print(lay_thong_tin_san_pham("iphone15")) # Lần 1: Chậm
27 print("-" * 20)
28 print(lay_thong_tin_san_pham("iphone15")) # Lần 2: Nhanh (Lấy từ Cache)
```

### 3. Redis dùng khi nào?

- **Caching (Phổ biến nhất):** Lưu các dữ liệu hay truy cập (session đăng nhập, trang chủ, top sản phẩm) để giảm tải cho Database chính.
- **Bảng xếp hạng (Leaderboard):** Redis có cấu trúc Sorted Set cực mạnh để xếp hạng game thủ theo thời gian thực.
- **Đếm lượt view/like:** Tăng giảm biến số cực nhanh.
- **Pub/Sub đơn giản:** Chat room thời gian thực (tuy nhiên nếu cần tin cậy cao thì nên dùng RabbitMQ).

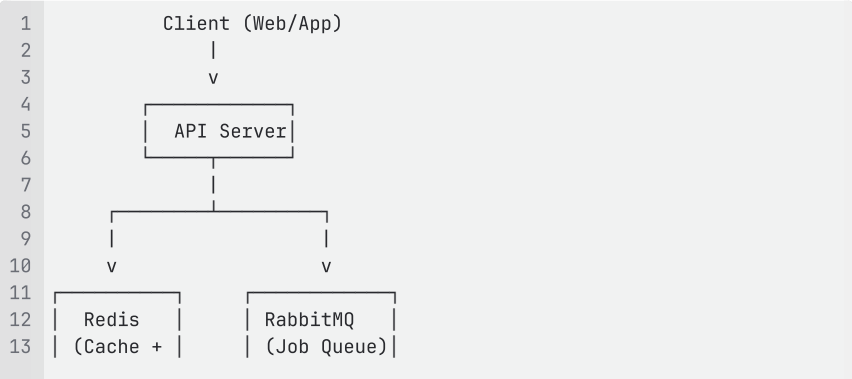
### PHẦN 3: TỔNG KẾT & SO SÁNH (KHI NÀO DÙNG CÁI NÀO?)

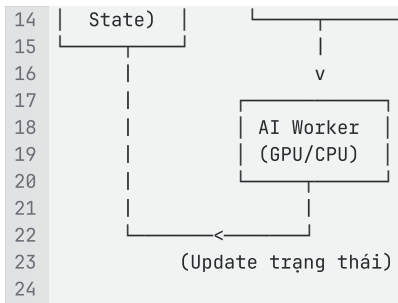
Đây là phần quan trọng nhất để các bạn không bị nhầm lẫn.

|               |  |   |
|---------------|--|---|
| Đặc điểm      | RabbitMQ (Người đưa thư)   | Redis (Sổ tay tốc độ cao)   |
| Vai trò chính | Message Broker (Chuyên chuyển tin)   | In-memory Store (Chuyên lưu & truy xuất nhanh)  |
| Lưu trữ       | Lưu trên ổ cứng (chủ yếu), xóa khi đã xử lý  | Lưu trên RAM (chủ yếu)  |
| Độ tin cậy    | <b>Rất cao.</b> Đảm bảo tin đến nơi, có xác nhận.  | Thấp hơn (trong vai trò message broker). Ưu tiên tốc độ.                              |
| Dữ liệu lớn   | Tốt cho việc điều phối quy trình phức tạp.   | Tốt cho dữ liệu nhỏ, truy xuất nhiều lần.   |
| Khi nào dùng? | Khi cần <b>kết nối</b> các hệ thống, xử lý tác vụ <b>chậm/nặng</b> (gửi mail, render video, xử lý đơn hàng). | Khi cần <b>tốc độ</b> (Caching), đếm số, bảng xếp hạng, hoặc Pub/Sub đơn giản (chat). |

**Ví dụ thực tế kết hợp cả hai:**

- Khi người dùng mua hàng:
  - **Redis:** Dùng để lưu Giỏ hàng tạm thời (để truy xuất nhanh khi họ lướt web).
  - **RabbitMQ:** Khi họ bấm "Thanh toán", thông tin đơn hàng được đẩy vào RabbitMQ để hệ thống kho, hệ thống gửi email, và hệ thống vận chuyển lần lượt lấy ra xử lý mà không làm treo ứng dụng của người dùng.





```

1  ### docker-compose.yml
2
3  version: '3.8'
4
5  services:
6    rabbitmq:
7      image: rabbitmq:3-management
8      container_name: rabbitmq
9      ports:
10       - "5672:5672" # AMQP protocol port
11       - "15672:15672" # Management UI port
12      environment:
13        RABBITMQ_DEFAULT_USER: guest
14        RABBITMQ_DEFAULT_PASS: guest
15      volumes:
16        - rabbitmq_data:/var/lib/rabbitmq
17      healthcheck:
18        test: rabbitmq-diagnostics -q ping
19        interval: 30s
20        timeout: 10s
21        retries: 5
22
23    redis:
24      image: redis:alpine
25      container_name: redis
26      ports:
27        - "6379:6379"
28      command: redis-server --requirepass mypassword
29      volumes:
30        - redis_data:/data
31      healthcheck:
32        test: ["CMD", "redis-cli", "-a", "mypassword", "ping"]
33        interval: 30s
34        timeout: 10s
35        retries: 5
36
37  volumes:
38    rabbitmq_data:
39      driver: local
40    redis_data:
41      driver: local
42

```

🟢 link code demo [GitHub - datbui-knm/redis\\_rabbitmq](https://github.com/datbui-knm/redis_rabbitmq)

Hy vọng bài giảng này giúp các bạn nắm rõ bức tranh toàn cảnh về RabbitMQ và Redis! Các bạn hãy thử chạy đoạn code demo trên máy để hiểu rõ hơn nhé. Có câu hỏi nào không?